Первое практическое домашнее задание.
Дедлайн: 19.02.2023 23:59:59

Требования к графу.

Так как граф это математическая абстракция, его можно параметризовать как угодно. Более того, это множество вершин и пар вершин aka рёбер.

1. Сделать абстрактный класс граф. Параметризовать его типом вершины (по умолчанию это ее номер) и типом ребра (по умолчанию это пара вершин).
2. От графа отнаследовать две реализации: на списке смежности и на матрице смежности.
3. Граф должен уметь.
   - Отдавать число вершин и ребер.
   - Отдавать по вершине список соседей (в вершинах).
   - Отдавать по вершине итератор на список соседей. Должен работать `range-based for`.
   - (По желанию) Сделать аналог `filter iterator`, который будет способен фильтровать ребра в зависимости от условия.
4. Прописать юзинги `vertex_type`, `edge_type`.

Требования к взвешенному графу.

1. Все требования к графу, кроме требования про наследника в виде матрицы смежности. Хватит на основе списка смежности.
2. Ребро должно быть взвешенным, то есть хранить в себе вес. Его тип считаем тоже шаблоном, чтобы уметь обрабатывать как целые, так и вещественные числа.

Требования к задаче 2A.

1. В задаче требуется лишь уметь искать расстояния, однако необходимо написать визитор, который для каждой вершины вернет предка в дереве кратчайших путей. Также он должен хранить расстояния для каждой вершины.
2. Ваш алгоритм (удобно сделать классом) должен уметь принимать граф и визитор.

Требования к задаче 2H2.

В данной задаче необходимо реализовать алгоритм A*.

1. Решение должно работать для квадратного поля произвольного размера (задайте шаблоном, достаточно реализовать специализации для 3 и 4).

2. Необходимо реализовать паттерн Visitor (параметризуется типами вершин и ребер). Вот пример, на который будет опора: boost. Необходимо реализовать абстрактный визитор (вам понадобятся не все методы из документации). Надо будет реализовать его наследника.
3. Ваш алгоритм (удобно сделать классом) должен уметь принимать граф и визитор (и еще кое-что).
4. Заметим, что ваш граф на самом деле не будет хранить все ребра и все состояния, то есть вам нужен лишь метод получения по вершине ее соседей. Как это скомбинировать с предыдущей реализацией графа? Одно из решений — сделать наследника абстрактного графа, в котором лишние методы будут удалены.
5. Заметим, что у нас будет вершиной какое-то интересное состояние State. Для решения задачи вам понадобится уметь хешировать State. Заметим, что State кодируется 16-ю числами по 4 бита, что идеально влезает в `uint64_t`. Для больших размерностей можно использовать `std::bitset`.
6. A* принимает в себя эвристику. Так как половина из вас — будущие машинлернеры, пришло время познакомиться с определенными практиками. Так как машинлернинг это поиск оптимальных гиперпараметров между двухчасовыми чиллами, надо это автоматизировать, чтобы перерыв между чиллами был минимален.

   А именно, у нас будет абстрактная эвристика (она имеет оператор `()` от произвольной вершины). Далее необходимо реализовать наследников (одного не хватит) для конкретных типов вершин (в вашем случае это State). И финалом будет наследник `ConvexHeuristic`, который в конструкторе принимает два вектора: вектор абстрактных эвристик и вектор коэффициентов, тогда результатом будет выпуклая комбинация эвристик с заданными коэффициентами.
7. Отдельно отметим, что задача имеет решение только в половине